REMARKS

This is intended as a full and complete response to the Office Action dated August 4, 2004, having a shortened statutory period for response set to expire on November 4, 2004. Please reconsider the claims pending in the application for reasons discussed below.

In the specification, the paragraphs [0014] and [0015] have been amended to correct minor editorial problems.

Claims 1-26 are pending in the application. Claims 26 remain pending following entry of this response. Applicants submit that the amendments and new claims do not introduce new matter. Respectfully, applicants traverse the rejection.

Claims 1-26 stand rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Pat. No. 6,263,489 to *Olsen et al.* (hereinafter "*Olsen*") in view of U.S. Pat. No. 6,161,216 to *Shagam*. Applicants respectfully traverse the rejection.

Olsen discloses methods for debugging machine code that has "been subjected to an optimizing action, wherein the machine code may have been reordered duplicated, eliminated or transformed, so as not to correspond with the programs source code order." *Olsen*, Abstract. Embodiments of *Olsen* allow the execution of the program being debugged in the same sequence as the source code instructions. *Olsen* 4:12-32. That is, using an elaborate structure of "high water marks" and "low water marks" that identify code sequences out of order with source code statements, the methods of *Olsen* emulate a sequence of machine instructions corresponding to the sequence of statements in the source code. *Olsen* 5:8-23. Contrary to the assertion by the Examiner, however, *Olsen* does not disclose "a method for inserting breakpoints in a program being debugged." Rather, for *Olsen* to operate, a user must have already set a breakpoint in the code. *Olsen*, Abstract, 5:28-31, 12:1-2, 12:18-20, Figure 4A, Figure 5. Specifically, Olsen handles two situations arising during debugging and code execution. A first situation is where code sequences occuring before a breakpoint originate from source code statements that appear after the breakpoint. In this

situation, the breakpoints are not executed. *Olsen*, 5:24-37. The second situation is where code sequences appear after the breakpoint, but appear before the break point in the source code. In this case, the breakpoints are executed using an emulation process. *Olsen*, 5:24-37. Either way, *Olsen* requires the existence of a breakpoint *a priori*. Applicants', however, claim a method for inserting breakpoints after identifying decision points influencing program behavior.

Further, the Examiner asserts that *Olsen* discloses "determining which blocks control execution of the basic block" where it discloses:

> Debugger 30 next determines whether a path exists from the machine code's entry point to machine code instruction "i" which does not contain a HW (decision step 108). If no (which means every path from the machine entry point to machine code "i" contains a HW), it proceeds to the next basic block (step 118) and continues at step 104.

Olsen, 12:40-45. This passage, however, discloses a step from a method that ensures that the sequence of machine code executed during debugging corresponds to the sequence of source code statements. In other words, from the first machine code, this method step determines whether any possible sequence exists to reach the machine code "i" without encountering a "high water mark" (HW). If encountered, the high water mark indicates the presence of machine instructions from a region of source code that appears subsequent to the breakpoint, and the sequence of execution of machine code statements is reordered accordingly "to execute any those instructions that correspond to source constructs preceding P, and emulates only those instruction," until a low water mark is reached. *Olsen*, 5:31-34. "Thus, when the user sets a source breakpoint at P, the debugger actually sets a machine code breakpoint at HW." *Olsen*, 5:29-31. Accordingly, *Olsen* discloses a method for reordering the breakpoints of the machine code to correspond to breakpoints selected for points within the sequence of statements of the source code, and not determining which blocks control execution of the basic block as claimed by applicants.

Finally, the Examiner concedes that *Olsen* does not expressly disclose "inserting a breakpoint at each branch contained in the blocks controlling execution of the basic

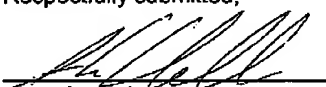
block." The Examiner asserts, however, *Shagam* discloses this limitation. The combination of *Shagam* and *Olsen*, however, simply fails to teach show or suggest a method for inserting breakpoints in a program being debugged as claimed by Applicants. Because *Olsen* fails to disclose the claimed step of determining which blocks control execution of the basic block, the techniques disclosed in *Shagam* lack the information necessary to determine where to insert any breakpoints into the program being debugged. Thus, the combination is inoperable.

For the foregoing reasons, Applicants submit that claims 1, 12, and 16 are patentable over *Olsen* in view of *Shagam*. Further, because claim 2-10, 13-15, and 17-23 depend from claims 1, 12, 16 respectively, Applicants submit that these claims are patentable over *Olsen* in view of *Shagam*.

The secondary references made of record are noted. However, it is believed that the secondary references are no more pertinent to the Applicants' disclosure than the primary references cited in the office action. Therefore, Applicants believe that a detailed discussion of the secondary references is not necessary for a full and complete response to this office action.

Having addressed all issues set out in the office action, Applicants respectfully submit that the claims are in condition for allowance and respectfully request that the claims be allowed.

Respectfully submitted,



Gero G. McClellan
Registration No. 44,227
MOSER, PATTERSON & SHERIDAN, L.L.P.
3040 Post Oak Blvd. Suite 1500
Houston, TX 77056
Telephone: (713) 623-4844
Facsimile: (713) 623-4846
Attorney for Applicants